ECE385

Fall 2021

Experiment 9

SOC with Advanced Encryption Standard in SystemVerilog

Ge Yuhao, Lou Haina
D231, Dec.16, 2021
TA: Chenhao Wang

1 Introduction

In this lab, we implemented an 128-bit AES encryption and an decryption module. The encryption part is done in software and the decryption part of the lab is done in hardware. So the encoding process is on the NIOs II system and the decoding process is on the hardware logic.

2 Written Description and Diagrams of the AES encryptor/decryptor

2.1 Written description of the software encryptor

General description First the console will do the communicating work between the system and the user, it will give hint to the user to type in the message to be encrypted and the key to use. Then the C program in the NIOS II processor will calculate the encrypted message and show that on the screen.

NIOS II processor The NIOS II system is used to run all the C code on hardware. With a JTAG UART module, it can also do the scanf function to communicate with our computer console. User can type message to be encoded into the console to let the system do encryption work. Also, the processor also communicates to the avalon bus which allows the transition of the encrypted and decrypted message between the hardware and the software.

Basic functionality Software has subBytes, AddRoundKey, Shiftrows a total of 8 small c help functions which help encryptor to encryte the message with a given key using CPU on NIOS II. we input the address of our states which stored in memory into our functions, and by calling functions in main function: encryption, we modify the value of states like adding it with around key using addroundkey function, shifting the three rows in shiftrows function, mixing its columns with special algrithoms learned in encryption and so on. Using keyexpansion function, we generate 10 around keys at once. The main function will accept messages and keys from users from console by scanf and encrypt it then output the encryption message to the console to user.

2.2 Written description of the hardware decryptor

Describe the basic steps of decryption and how this is controlled and computed in hardware

General description The hardware use logic to compute the decrypted message basing on the given encripted message and key. A FSM is also used to control the whole process.

Basic steps We have small mocules like InvMixColumns, InvShiftRows, KeyExpansion and InvSubBytes modules. We instantiated all these modules, and used a finite state machine to combine those modules to do the decryption. The round keys are generaed in the KeyExpansion, and the 9th round key is used at first. A counter is used in the FSM to keep track of the round number. Each round consists of InvShiftRows, InvSubBytes, AddRoundKey and InvMixColumns, and this will loop 9 times. After the loop, we perform InvShiftRows, InvSubBytes and AddRoundKey one last time then we can get the final decrypted message.

2.3 Written description of the hardware/software interface (avalon_aes_interface.sv)

The avalon_aes_interface module is used to connect the hardware and the software. There are 16 32-bits registers in this module to do the connection work. After the NIOS II is done with encryption, it will store the key and the encrypted message in to the first 4 registers and registers 4-7, so they can be used by the hardware.

Similarly, after the decryption module is done with the process, the decrypted message is stored into registers 8-11 for the software to read. Register 14 and 15 is used to tell whether the hardware should start the decryption process and whether the decrypting work is done by the software. The unused registers 12 and 14 can be used to sent other data. Any other data is sent through the unused registers, registers 12 and 13.

2.4 Block diagram

Please include the RTL view of avalon_aes_interface.sv. The Qsys view of the NIOS processor or lab9_top.sv is not necessary for this portion.

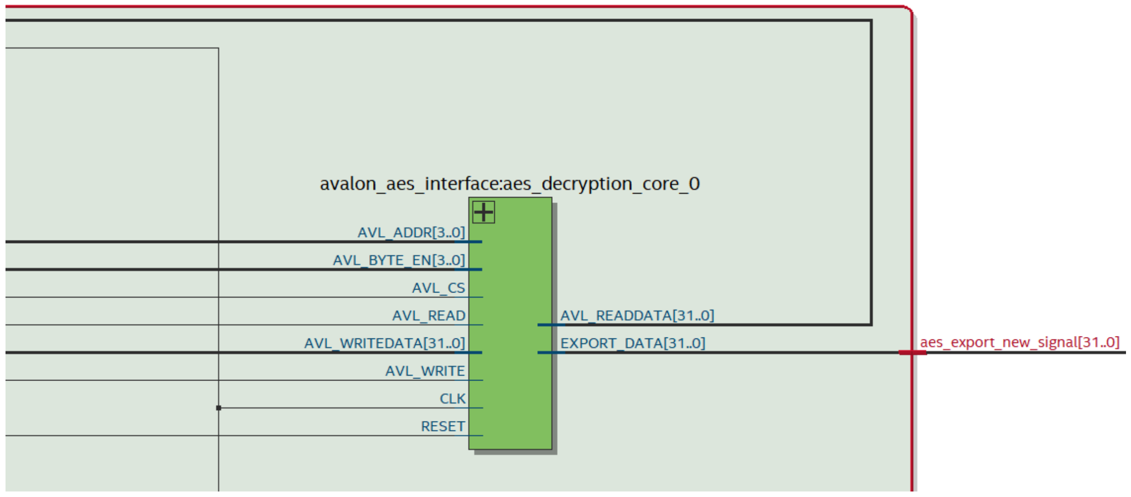


Figure 1: The RTL view of avalon_aes_interface.sv

2.5 State Diagram of AES decryptor controller

This is the state machine that was written in AES.sv. You may abbreviate the 9 looping rounds in the state diagram like in figure 9 on page IAES.9 of the lab manual

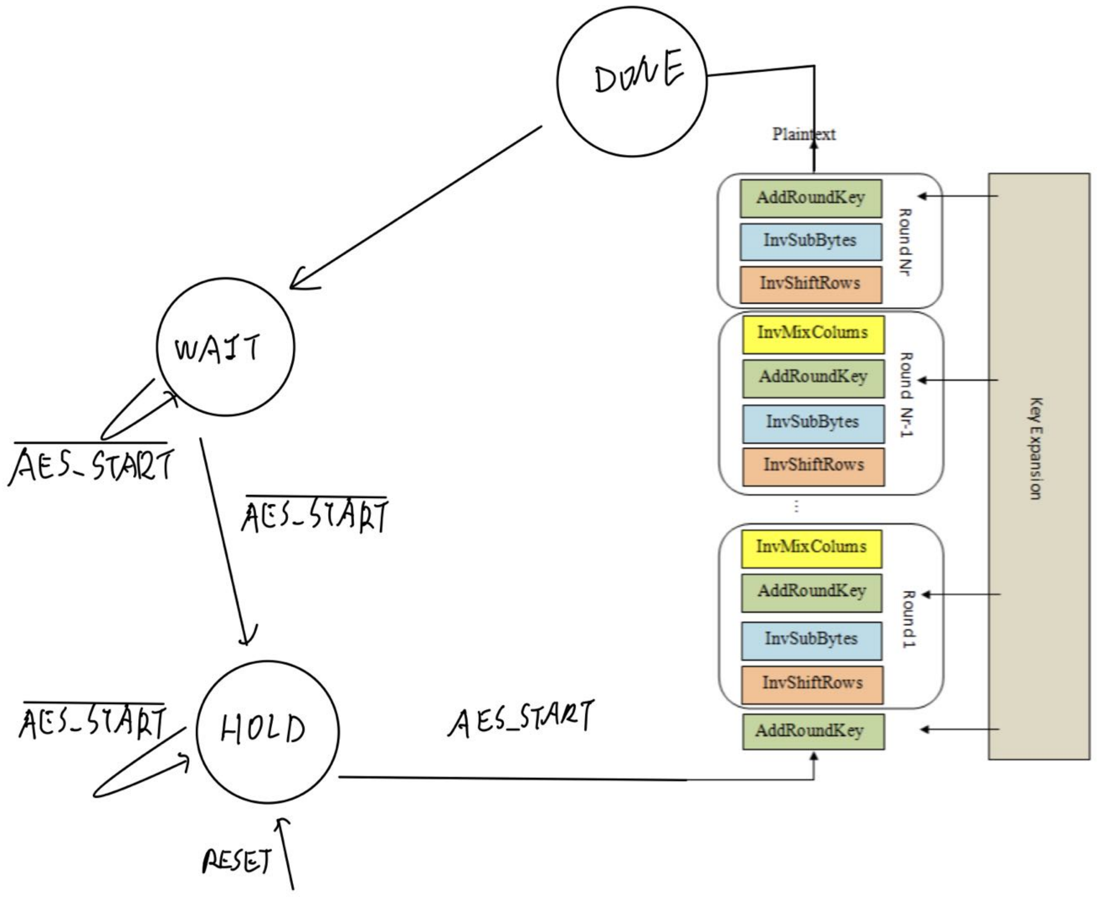


Figure 2: State Diagram of AES decryptor controller

2.6 Module Descriptions

A guide on how to do this was shown in the Lab 5 report outline. Do not forget to describe the Qsys generated file for your Nios system!
Module: InvAddRoundKey
Inputs: state, key
Output: result
Description: This module XORs the input message and the round key.
Purpose: This module is used in the decryption process where we used it to subtract(XOR) the round key from the message.

Module: AES
Inputs: CLK, RESET, AES_START, AES_DONE, AE_KEY, AES_MSG_ENC
Output: AES_MSG_DEC
Description: This module uses a state machine to control the entire devryption process.
Purpose: This module connect all the other modules, and it controls the data coming from several different decryption modules.

Module: Avalon_aes_interface
Inputs: CLK, RESET, AVL_READ, AVL_WRITE, AVL_CS, AVL_BYTE_EN, AVL_ADDR, AVL_WRITEDATA
Output: AVL_READDATA, EXPORT_DATA
Description: This module creates a register file of 16 registers with each one having a different purpose.

Purpose: This module is a linker between the software and hardware process of this lab, which enables the communocation between the software and hardware.

Module: HexDriver
Inputs: IN
Output: OUT
Description: This displays the input to the module to the hexdisplays on the DE2 board.
Purpose: This module helps us to read the data from hardware, this can be used for debugging purpose.

Module: InvMixColumns
Inputs: IN
Output: OUT
Description: This module does inverse multiplication on 1 column of decrypted message.
Purpose: This module serves as part of the decryption algorithm. It conducts the inverse Mixcolumn on the hardware.

Module: InvShiftRows
Inputs: data_in
Output: data_out
Description: This module does inverse shift rows operations with the input data.
Purpose: This module serves as part of the decryption algorithm. It conducts the inverse shift row on the hardware.

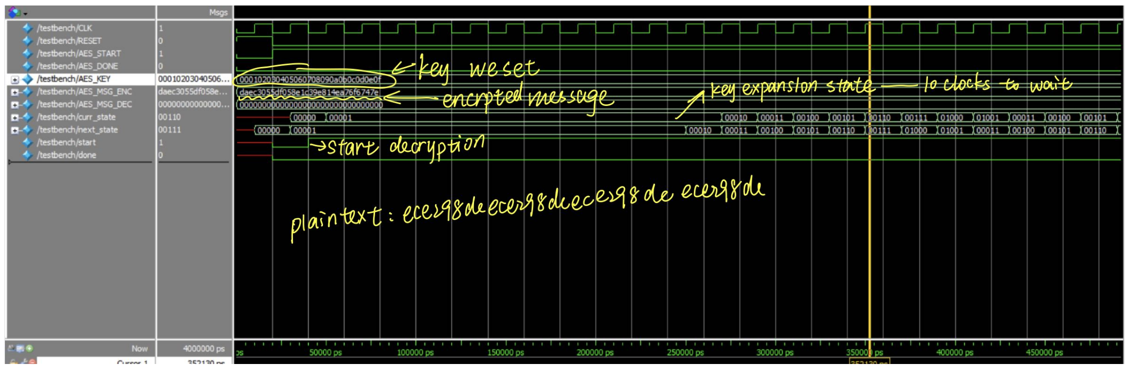
Module: KeyExpansion
Inputs: Cipherkey
Output: KeySchedule
Description: This module generated the round key used in the decryption algorithm.
Purpose: This module serves as part of the decryption algorithm. It conducts the KeyExpansion on the hardware.

Module: lab9_top
Inputs: CLOCK_50, KEY
Output: LEDG, LEDR, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7, DRAM_ADDR, DRAM_BA, DRAM_CAS_N, DRAM_CKE, DRAM_CS_N, DRAM_DQ, DRAM_DQM, DRAM_RAS_N, DRAM_WE_N, DRAM_CLK
Description: This module is the top-level of this lab.
Purpose: We use this module to connect all the corresponding wires of hexdrivers and sdram.

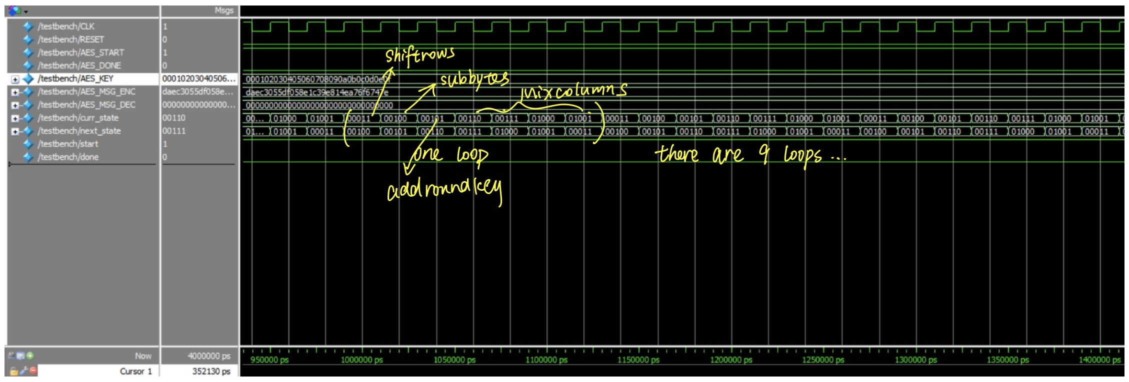
Module: SubBytes
Inputs: Clk, in
Output: out
Description: This moduled helps to transform the input bytes.
Purpose: This module serves as part of the decryption algorithm. It e helps us to perform the transform in the Rijndael's finite field.

Module: Lab9_soc
Inputs: Clk_clk, reset_reset_n
Output: Aes_export_export_data, Sdram_clk_clk, Sdram_wire_addr, Sdram_wire_ba, Sdram_wire_cas_n, Sdram_wire_cke, Sdram_wire_cs_n, Sdram_wire_dq, Sdram_wire_dqm, Sdram_wire_ras_n, sdram_wire_we_n
Description: This module hold the NIOS2 chip logic.
Purpose: The NIOS II handles the software encryption part of the lab.

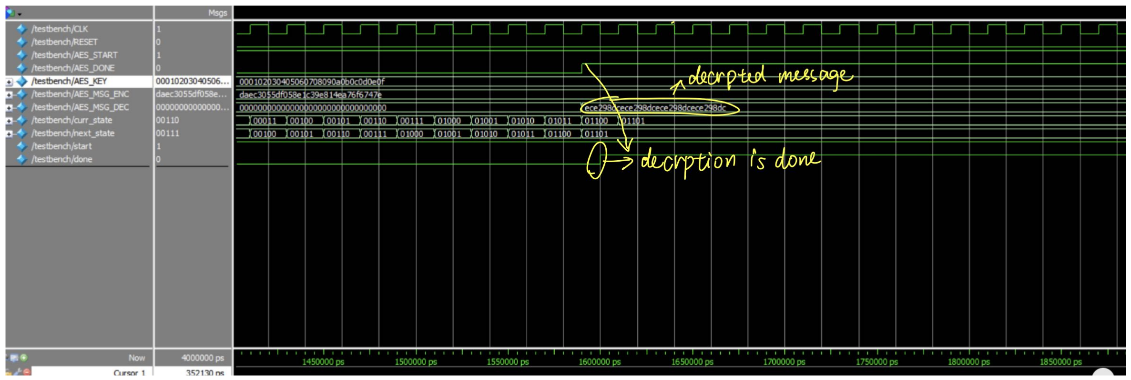
3 Annotated Simulation of the AES decryptor



(a) simulation1: decryption start



(b) simulation2: decryption process



(c) simulation3: decryption end

Figure 3: testbench simulation for AES.sv

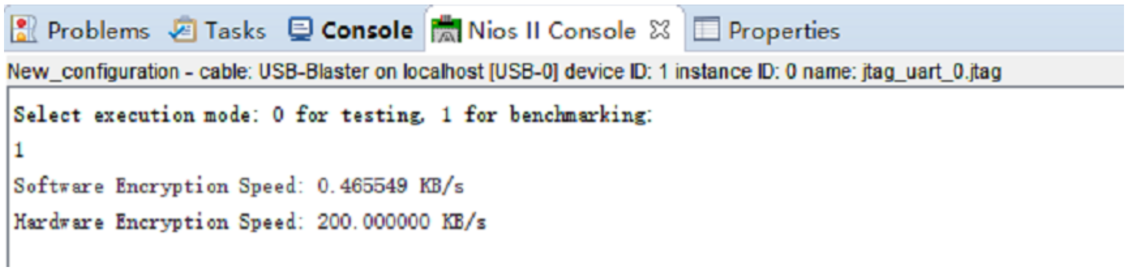
4 Post-Lab Questions

LUT	7249
DSP	0
Memory(BRAM)	125,952
Flip-Flop	2808
Frequency	135.17Mhz
Static Power	102.53mW
Dynamic Power	72.29mW
Total Power	242.91mW

Table 1: Design Resources and Statistics

4.1 Which would you expect to be faster to complete encryption/decryption, the software or hardware? Is this what your results show? (List your encryption and decryption benchmark here)

We think that the hardware would be faster, and the result consists with our prediction. The software's speed is 0.465549 KB/s while the hardware speed is 200 KB/s



```
New_configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instance ID: 0 name: jtag_uart_0.jtag
Select execution mode: 0 for testing, 1 for benchmarking:
1
Software Encryption Speed: 0.465549 KB/s
Hardware Encryption Speed: 200.000000 KB/s
```

Figure 4: The benchmark output

4.2 If you wanted to speed up the hardware, what would you do? (Note: restrictions of this lab do not apply to answer this question)

We can use the hardware more paralleled to conduct the decoding process. For now, we calculated all the key expansion at the same time, then do other thing when this is finished. To make it faster, we can rearrange the structure of the state machine and move to later states while calculation the key expansion. Only need to guarantee that the first expanded key is ready when the state proceeds to next.

5 Conclusion

5.1 Discuss functionality of your design. If parts of your design didn't work, discuss what could be done to fix it

Our design works perfectly well, and the encryption speed of hardware is properly faster than software which confirms what we suppose.

5.2 Was there anything ambiguous, incorrect, or unnecessarily difficult in the lab manual or given materials which can be improved for next semester? You can also specify what we did right so it doesn't get changed.

Most of content in the material is very clear and helpful, but if you could include some explanations of the hardware implementation of decryptor, for example, how you implement the InvSubBytes.sv module? is it combinational? it will be better.